BLACK CODHER

CODING PROGRAMME

< CODING BLACK FEMALES />

Black Codher Bootcamp

NÍ YỌ̀

BLACK CODHER

CODING PROGRAMME

UNIT 3
Javascript 103

RECAP

- Operators
- Comparisons
- Conditional Logic

WHAT YOU'LL BE LEARNING TODAY?

BLACK CODHER

- While Loops
- For Loops
- Arrays
- Arrays WITH Loops

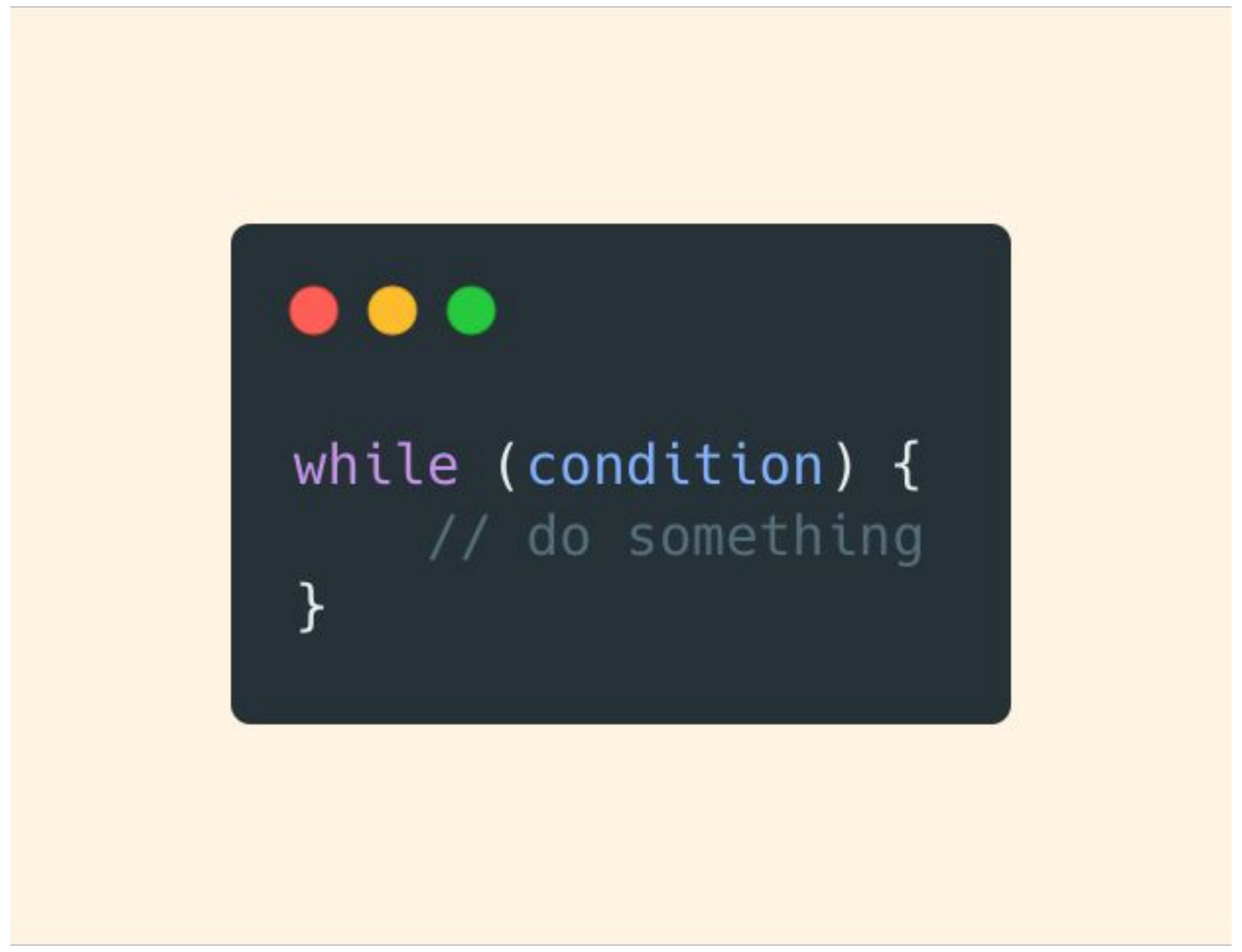
LOOPS

WHAT IS A LOOP?

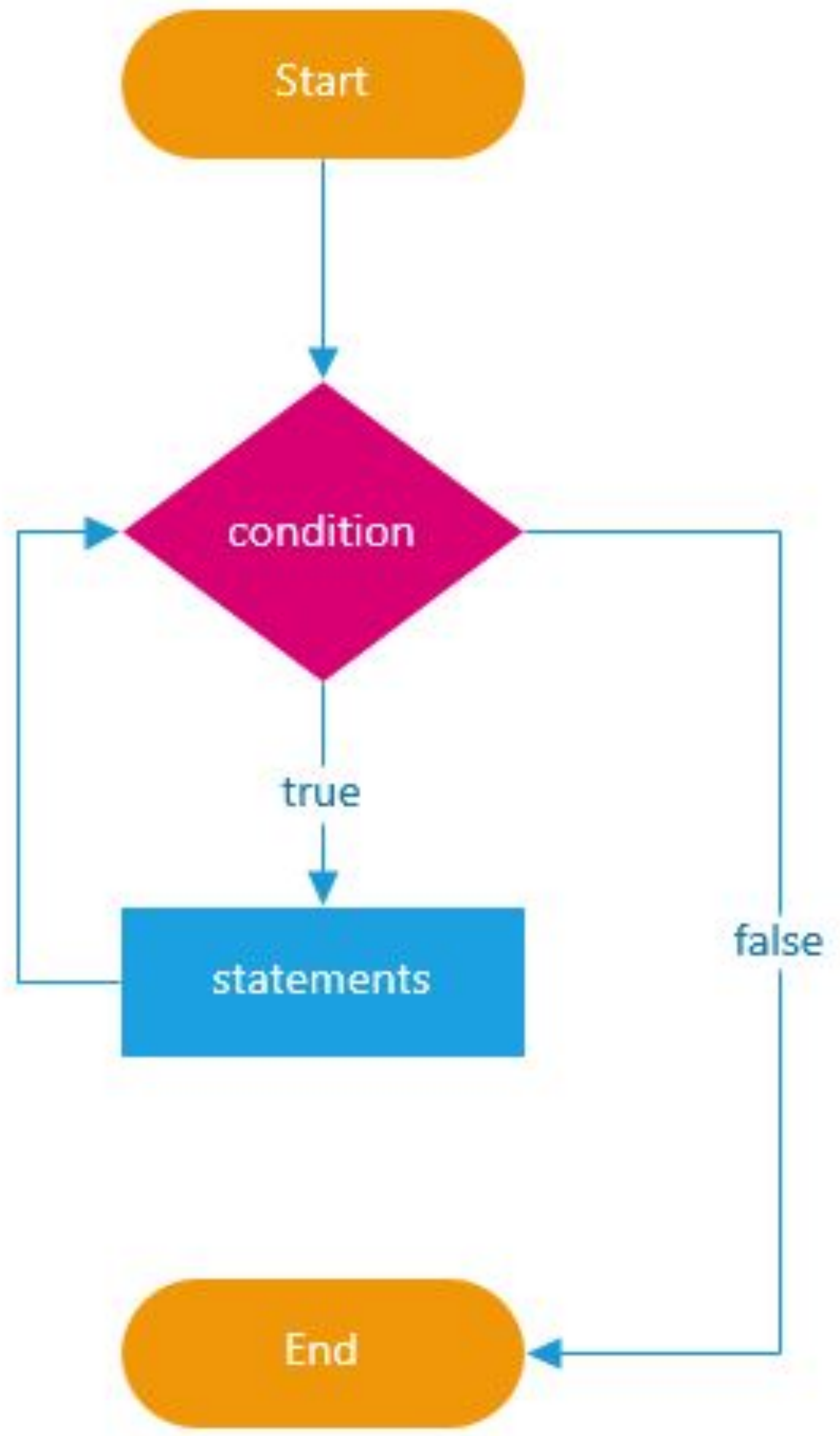
A loop is used to execute a set of statements repeatedly until a condition is met.

WHILE LOOP

The `while` loop is the simplest of all of the loops. It looks a bit like an `if` statement and the content in the parentheses `()` is the condition that needs to be met so that the body of the function begins to execute:


```
while (condition) {
    // do something
}
```

FLOW OF A WHILE LOOP



WHILE LOOP VS IF STATEMENT

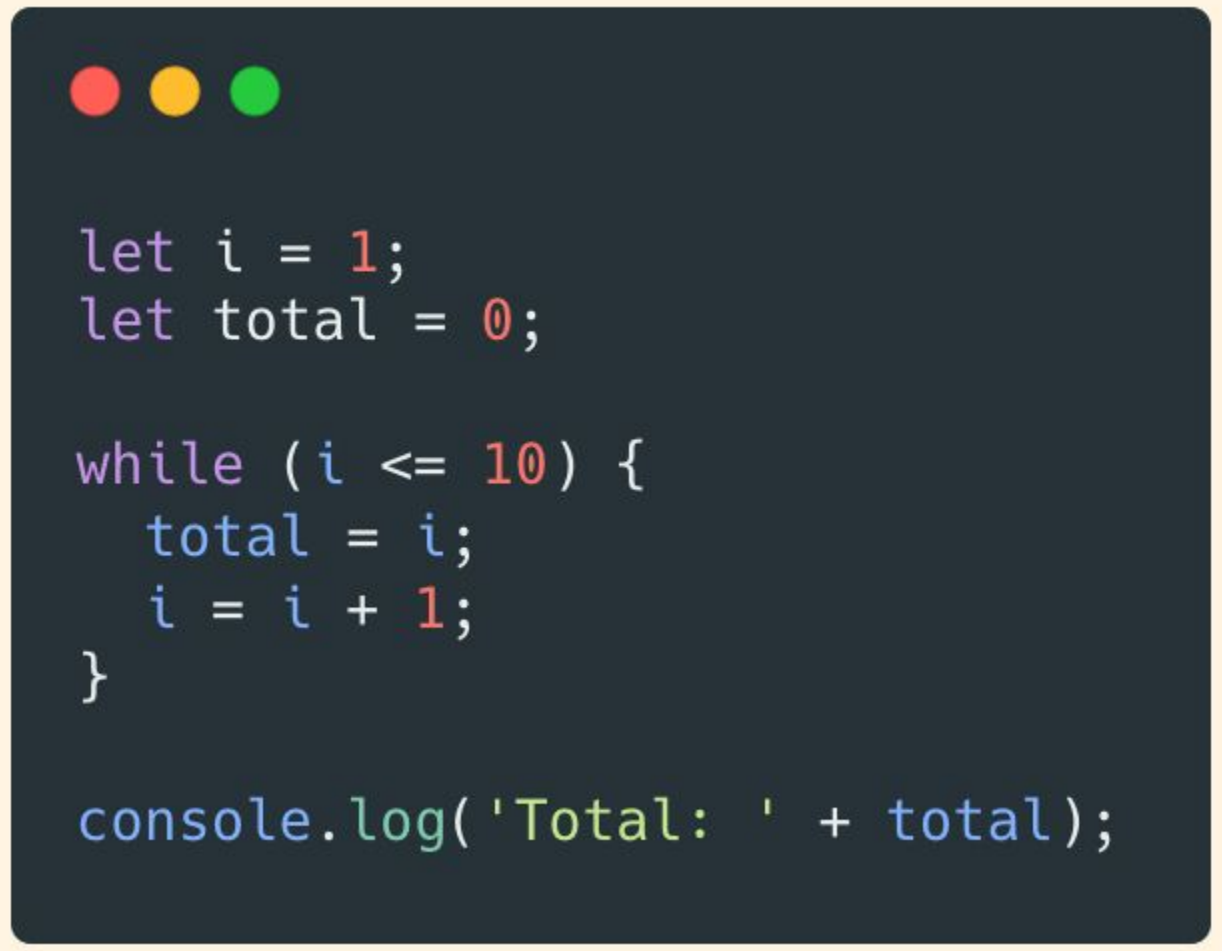
The difference between a `while` loop and an `if` statement is what happens after the statements in the block.

With an `if`, everything is finished and the statements below the if block are executed. With a while, we go back up to the condition.

If the condition is still true, the statements in the block are executed again, and so on until the condition becomes false. This is why we call it a loop.

SHOE SHOP EXAMPLE

If we wanted to set a limit on the number of shoes a customer could place in their basket, we could count to the shoe limit and while it hasn't been reached, the option to buy more shoes is still be available.



```
let i = 1;
let total = 0;

while (i <= 10) {
  total = i;
  i = i + 1;
}

console.log('Total: ' + total);
```

SHOE SHOP EXAMPLE BREAKDOWN

- Starts with variables `i = 1` and `total = 0`
- While `i` is less than or equal to 10
- Add 1 to `i`, and update `i` with the result (increment `i`).

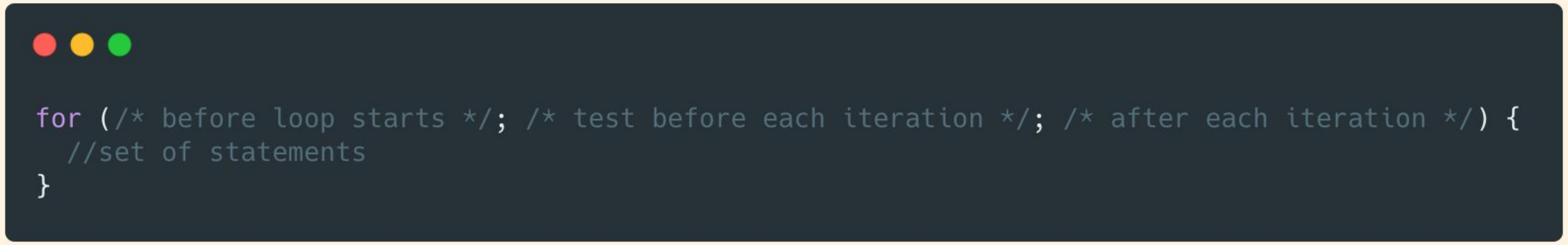
TASK

Create a `while` loop that prints a random number to the console whilst our count is less than 10.

There is already a `randomNumber` variable in the `index.js` (see the *session3* folder) that you can use inside the `while` loop to create the random number that should be printed.

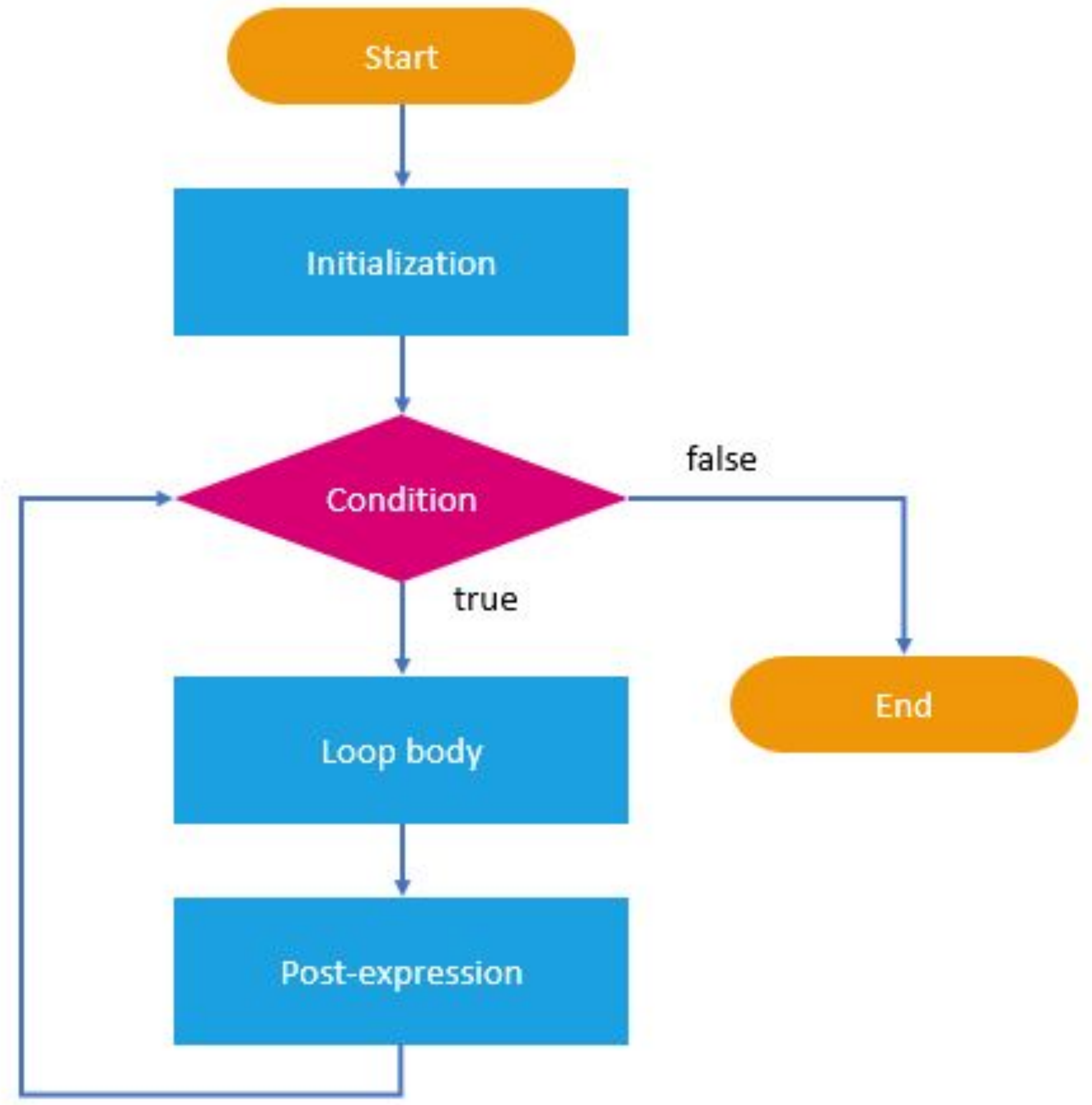
FOR LOOP

The `for` loop offers the same behaviour as a `while` loop, but arranged in a way that is often more convenient. It's very common in loops to have a counter (as there was in the case above), and the `for` loop caters especially for this.



```
for (/* before loop starts */; /* test before each iteration */; /* after each iteration */) {
  //set of statements
}
```

FLOW OF A FOR LOOP



FOR LOOP

The content in the parenthesis `()` is split into three parts by a `;`:

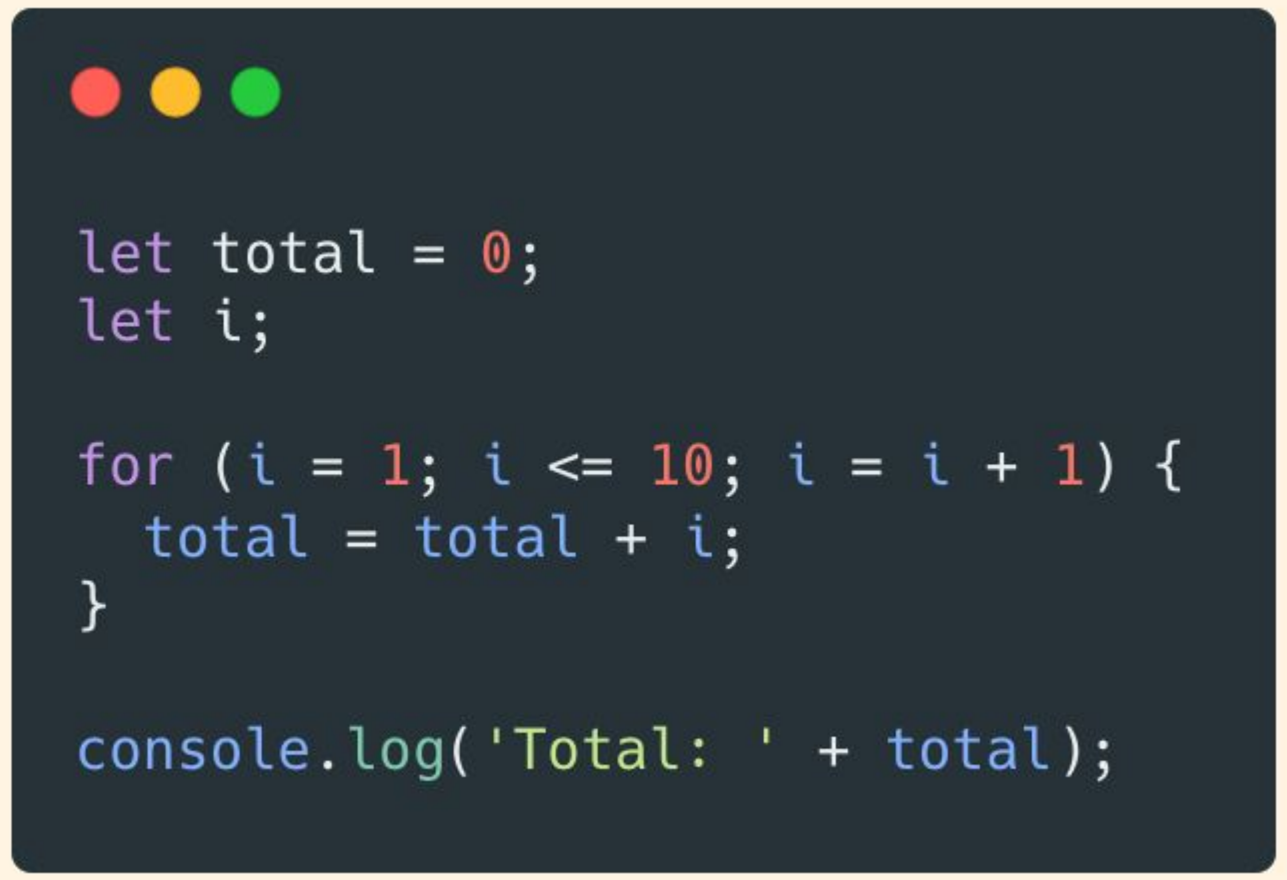
1. The first part is used once, before the loop begins. It's a good place to set an initial value for a counter (like `i = 1` in the `while` loop example).
2. The second part is a test, and just like in the `while` loop it is checked before each iteration.
3. The third part is executed after each loop iteration. It's useful for incrementing the loop counter.

SHOPPING BASKET EXAMPLE

If we had an online shopping basket, we could loop over the items in the basket and add up the cost to a total using a `for` loop. The initial value can be zero, the second part can test that there are still items left to be added up and the third part can increment to the next item. The code that is run on each iteration can add the cost to a shopping basket total.

FOR LOOP

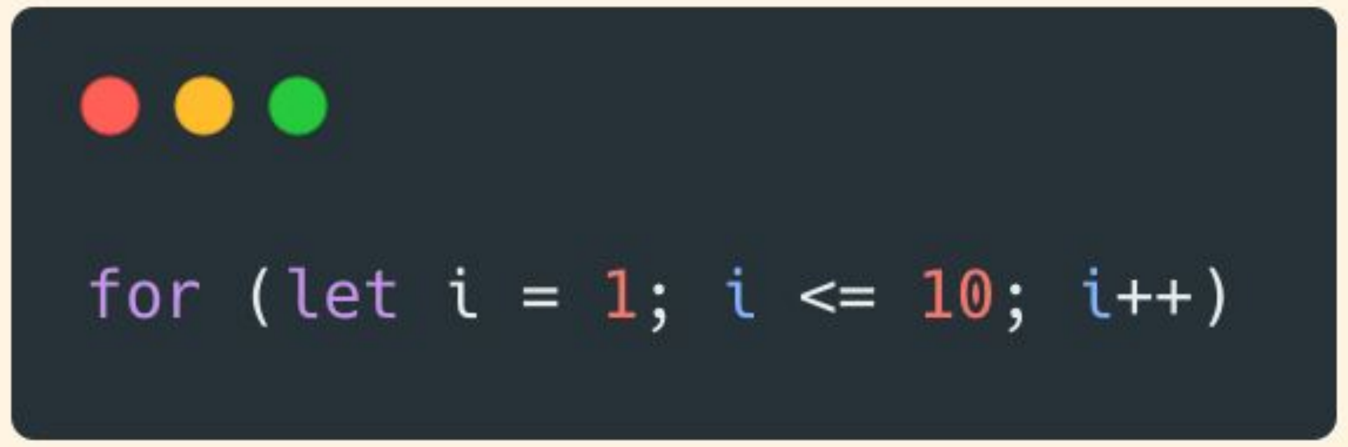
Let's rewrite the `while` loop as `for` loop:



```
let total = 0;
let i;

for (i = 1; i <= 10; i = i + 1) {
  total = total + i;
}

console.log('Total: ' + total);
```

FOR LOOP

We can also write a for loop with the **increment** operator:



```
for (let i = 1; i <= 10; i++)
```

The i++ is the shorthand of "increase i by one"

TASK

Rewrite the `while` loop from the previous task as a `for` loop.

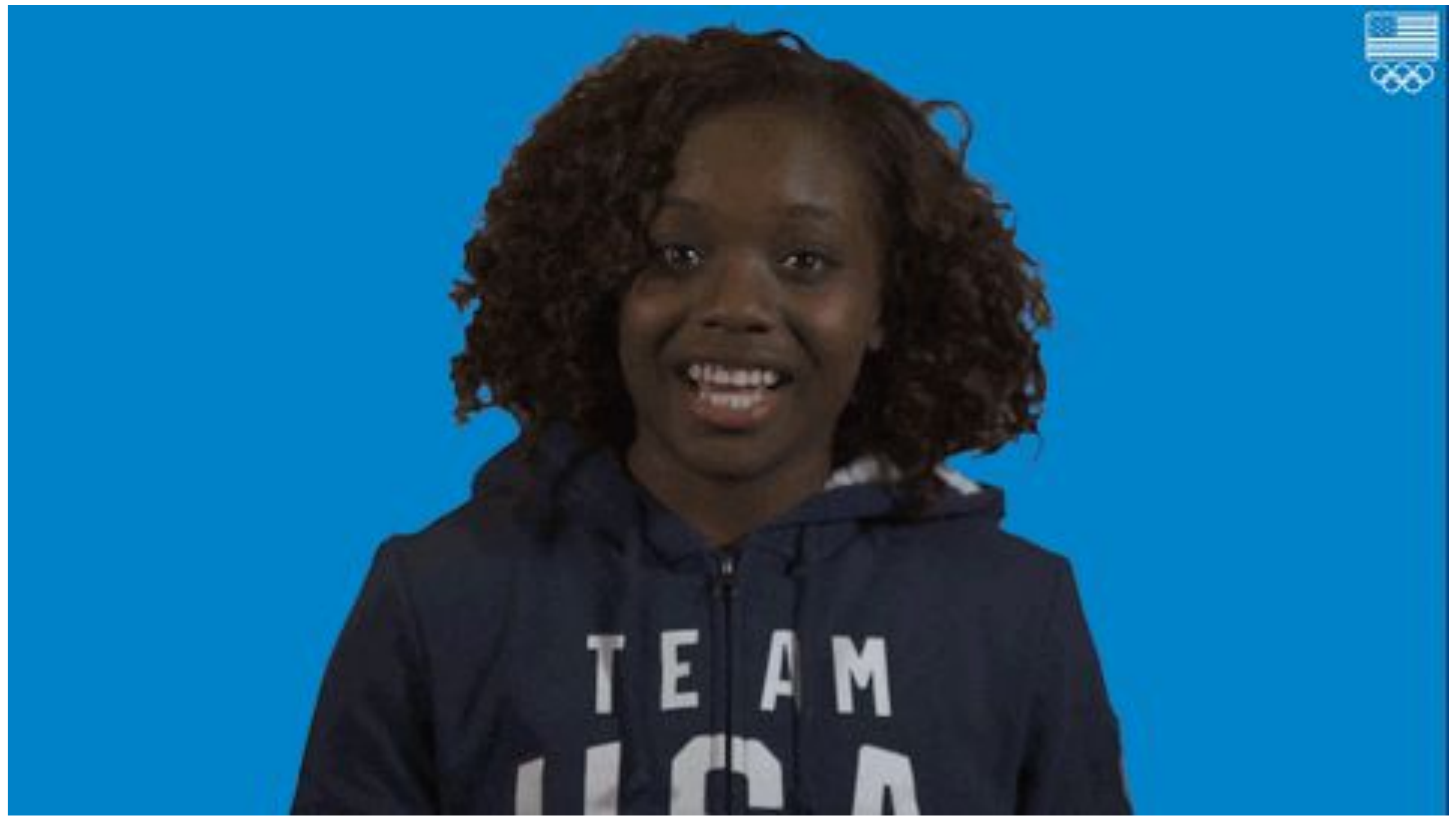
Again, you can use `randomNumber` variable in the `index.js` (see the *session3* folder) that you can use inside the `for` loop to create the random number that should be printed to the console.

FOR LOOP OR WHILE LOOP IRL

Even though while loops are more simple than for loops, it is more common to see for loops. This is because loops are often used to do something with arrays.

You might be wondering what an array is...well we'll be finding out soon.

WHAT YOU'VE LEARNED SO FAR



- What a `while` loop is
- When to use a `while` loop instead of an `if` statement
- What a `for` loop is

Checkpoint!

BLACK CODHER

How are you feeling?

RED - I have no idea what you're talking about

YELLOW - I have some questions but feel like I understand some things

GREEN - I feel comfortable with everything you've said

ARRAYS

WHAT IS AN ARRAY?

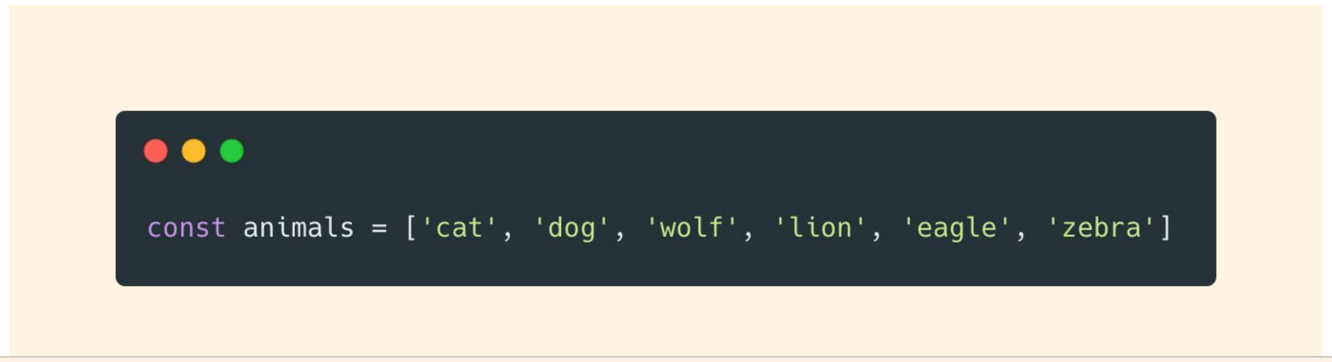
An **array** is a simple ***data structure*** and in Javascript is considered an object. It can hold a list of **elements** of the same or different types (e.g. strings, numbers, booleans, objects).

In an array each element can be accessed using the **index**.

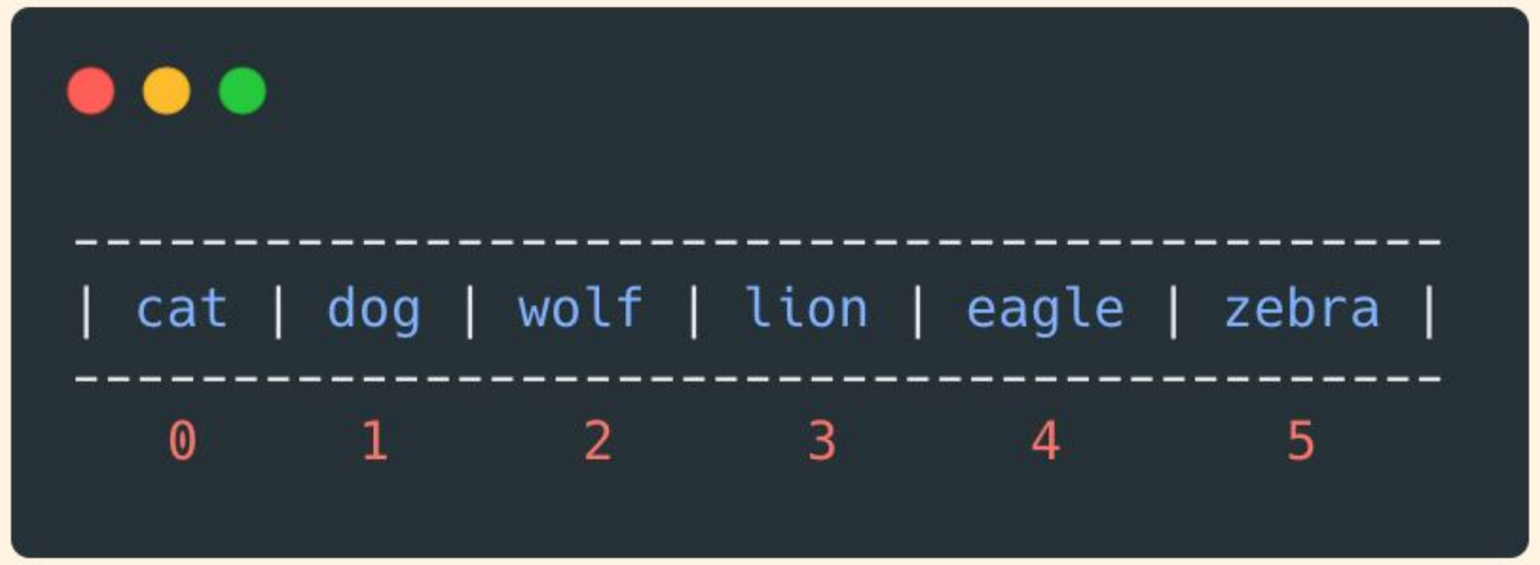
WHAT IS AN ARRAY?

Although confusing, it is important to remember that the first **index** of an array is `0`, not `1`.

Let's create an array of strings:



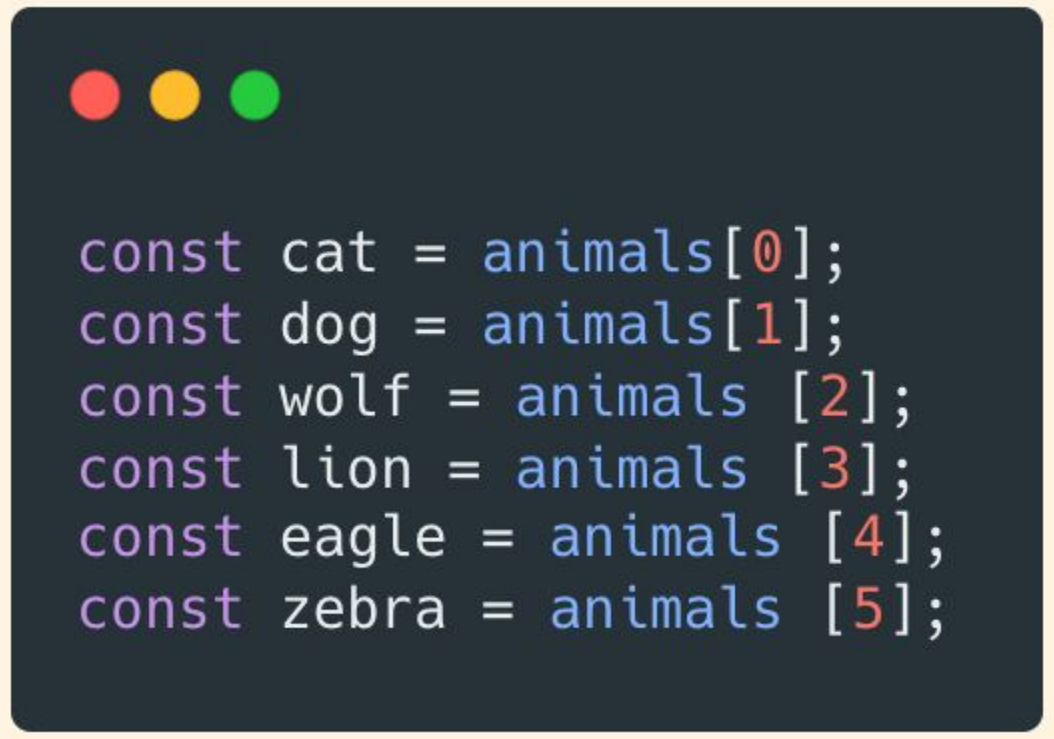
```
const animals = ['cat', 'dog', 'wolf', 'lion', 'eagle', 'zebra']
```



```
-------------------------------------------------
| cat | dog | wolf | lion | eagle | zebra |
-------------------------------------------------
   0     1     2      3      4       5
```

WHAT IS AN ARRAY?

To retrieve an item from the array, we use square bracket notation. So to retrieve the first item (`'cat'`) we use `animals[0]`, the second item (`'dog'`) is `animals[1]` and so on.

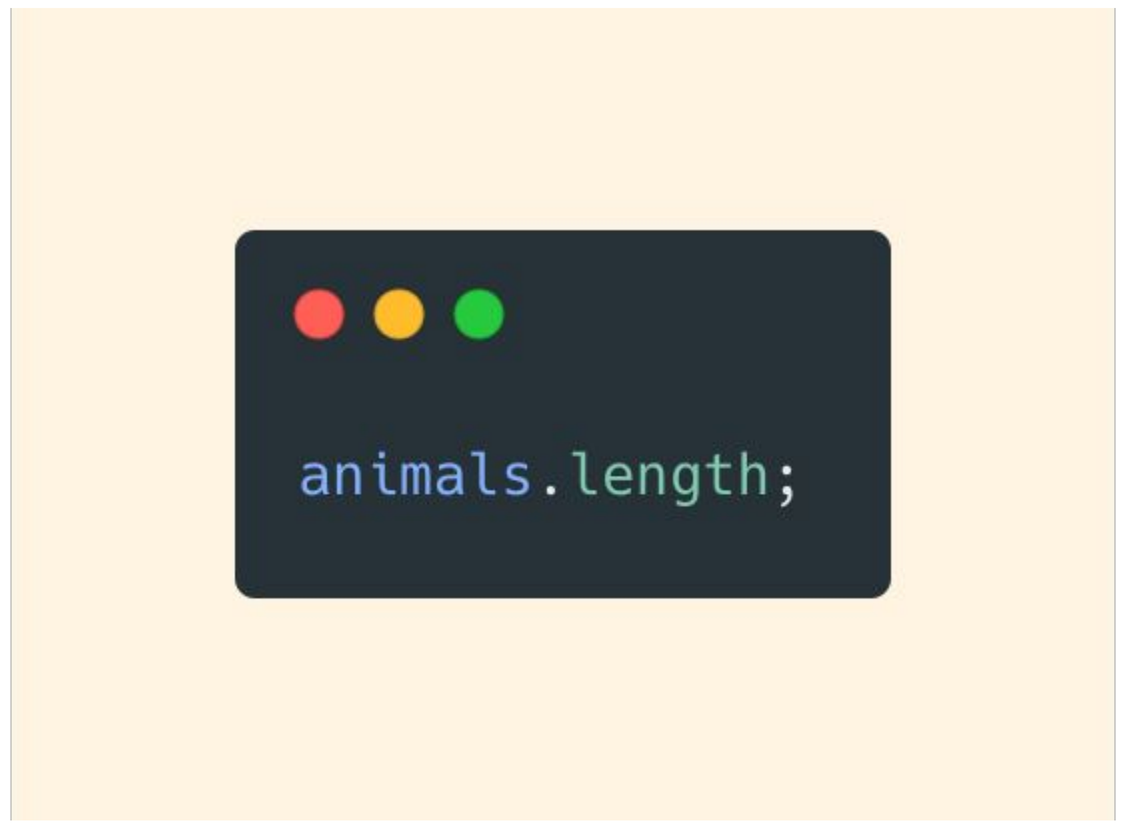


```
const cat = animals[0];
const dog = animals[1];
const wolf = animals [2];
const lion = animals [3];
const eagle = animals [4];
const zebra = animals [5];
```

LENGTH PROPERTY

The length property returns the number of elements in the array.

We can obtain the array length like so:

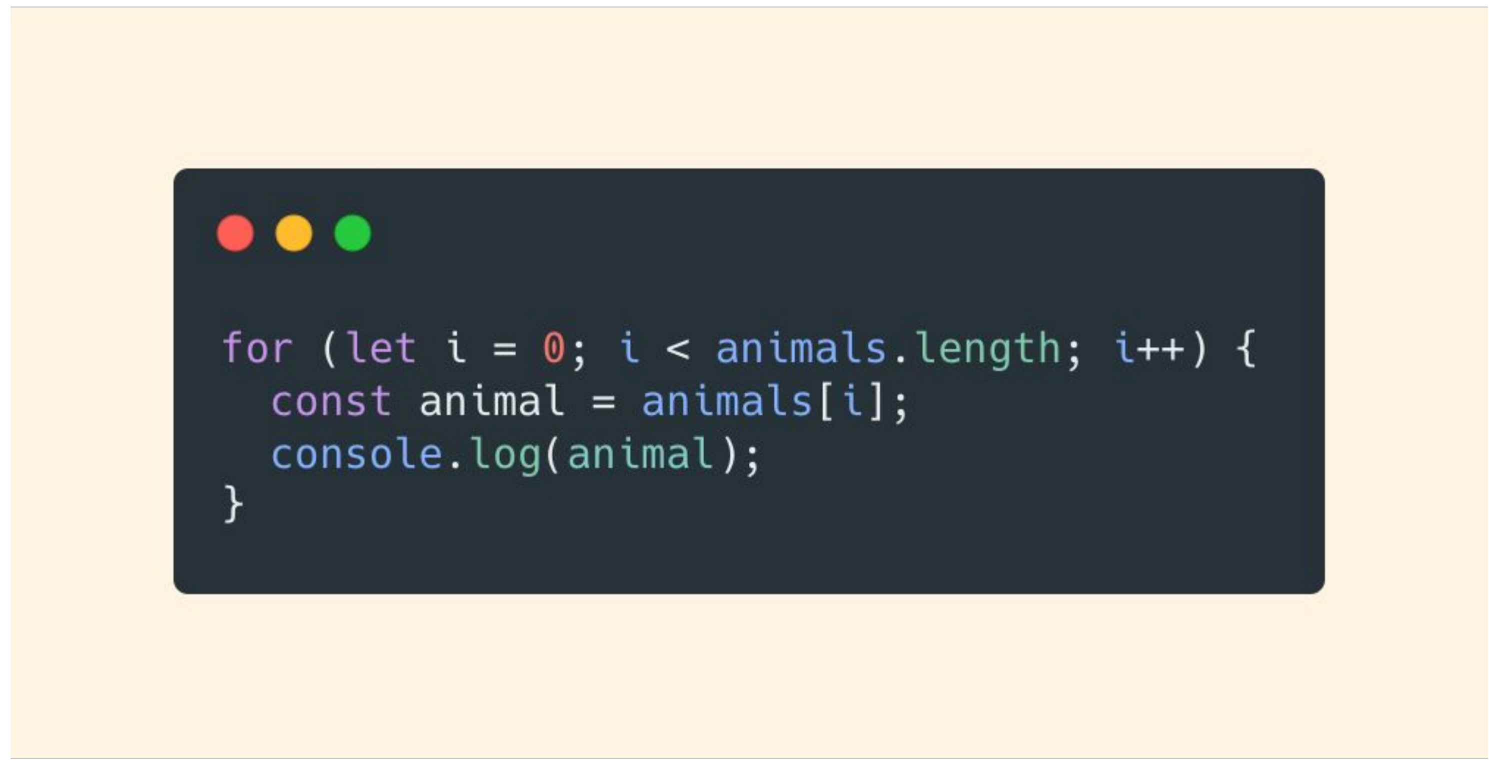


```
animals.length;
```

LENGTH PROPERTY

This is an extremely useful property and can help when you want to do something with every element in an array.

Let's `console.log` each animal of our `animals` array. We can use `animals.length` with a `for` loop:



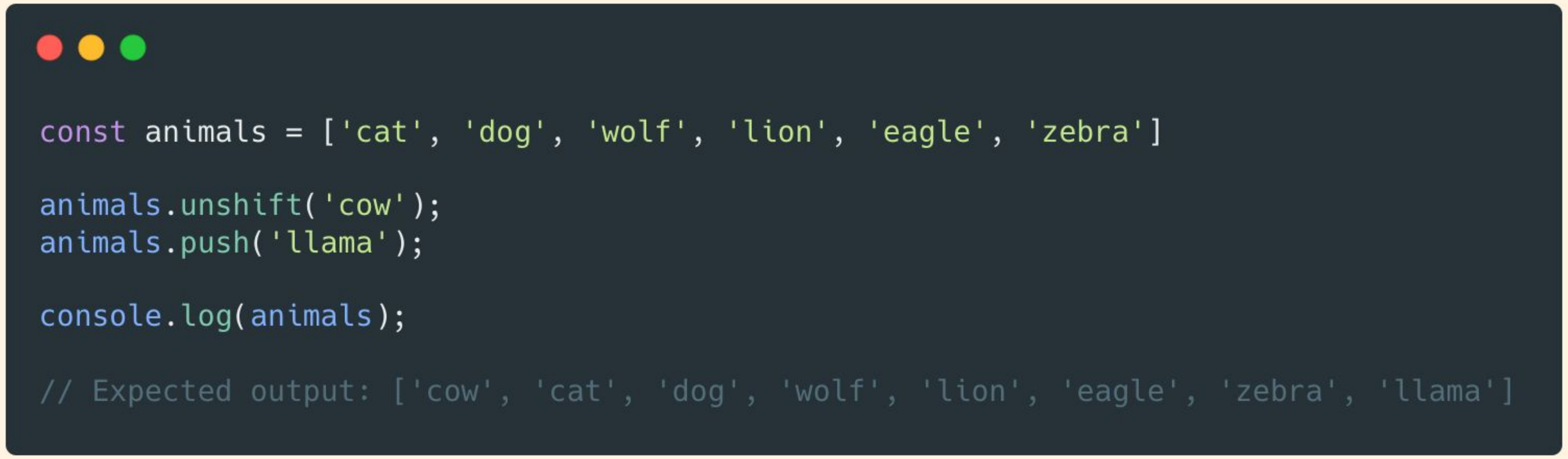
```
for (let i = 0; i < animals.length; i++) {
  const animal = animals[i];
  console.log(animal);
}
```

Note that we go up to, but do not include `animals.length` as an index. This is because arrays are indexed from zero, so the last index is always one less than the length.

ARRAY METHODS

ADDING TO AN ARRAY

- `array.push(element)` adds an element to the end of the array
- `array.unshift(element)` adds an element to the beginning of the array



```
const animals = ['cat', 'dog', 'wolf', 'lion', 'eagle', 'zebra']

animals.unshift('cow');
animals.push('llama');

console.log(animals);

// Expected output: ['cow', 'cat', 'dog', 'wolf', 'lion', 'eagle', 'zebra', 'llama']
```

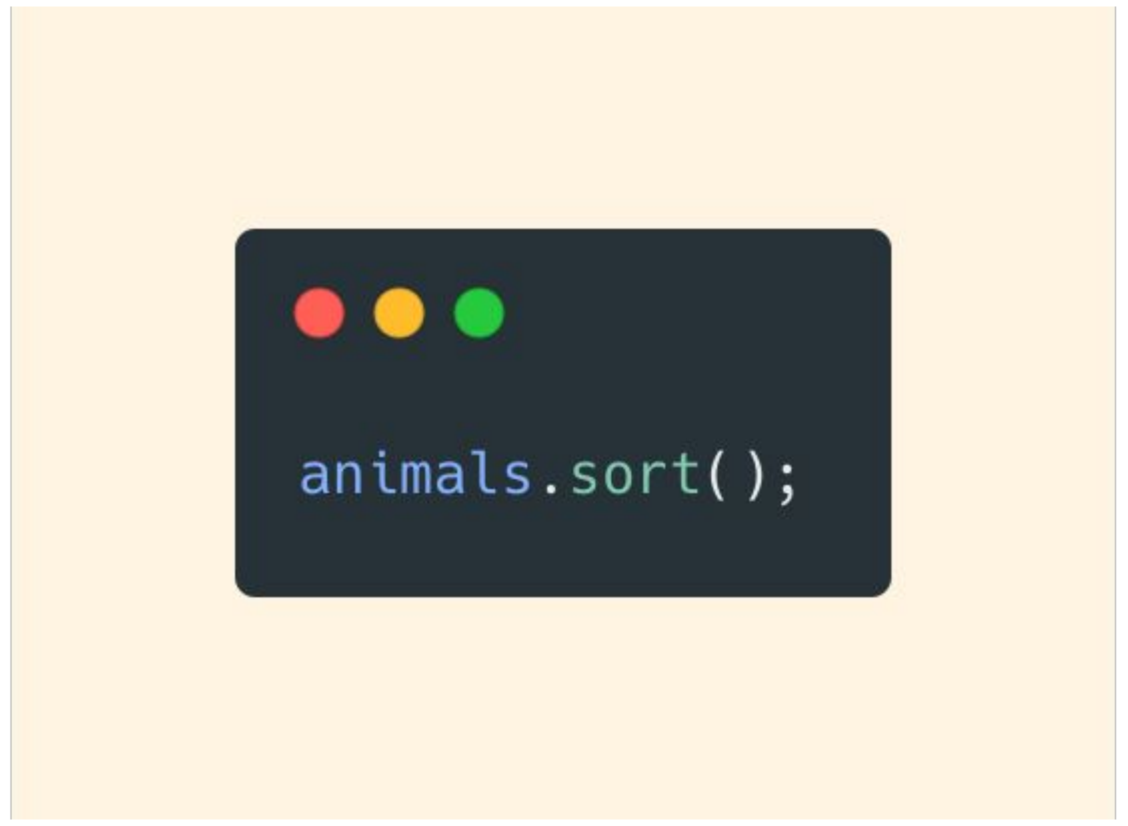
TASK

What do you expect to get when apply `pop()` to the animals array?

Try it out.

ORDERING ELEMENTS IN ARRAYS

To order the elements of an array we can use the `sort()` method.

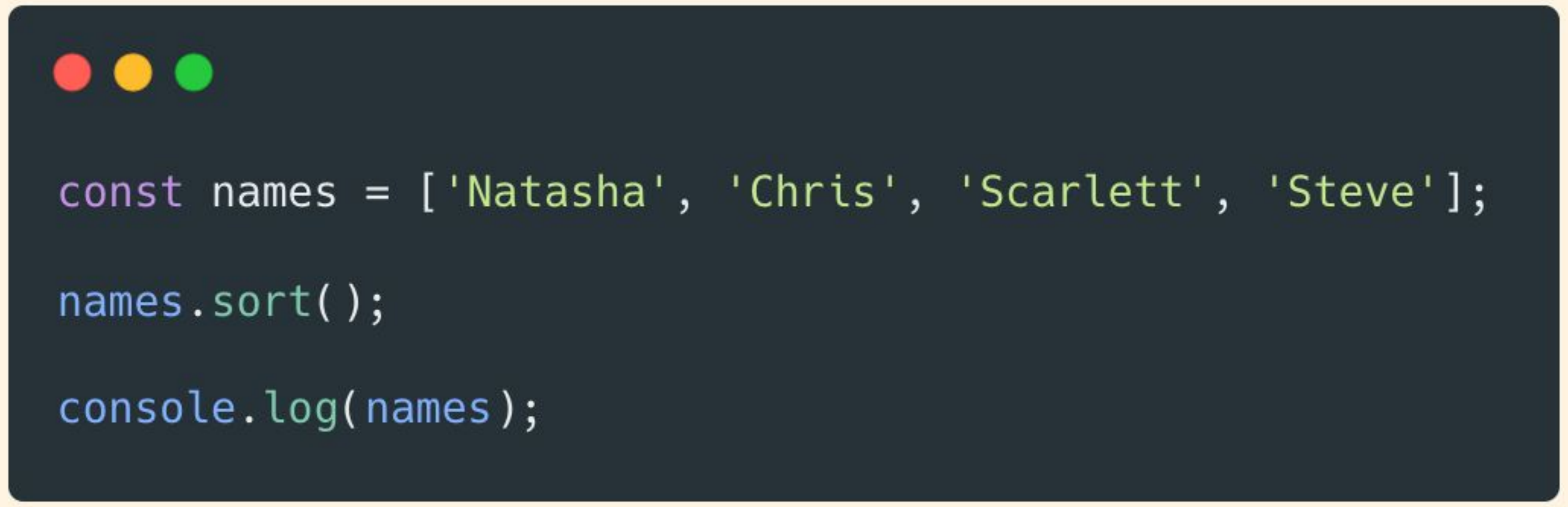


```
animals.sort();
```

ORDERING ELEMENTS IN ARRAYS

BLACK CODHER

Let's try it out:



```
const names = ['Natasha', 'Chris', 'Scarlett', 'Steve'];

names.sort();

console.log(names);
```

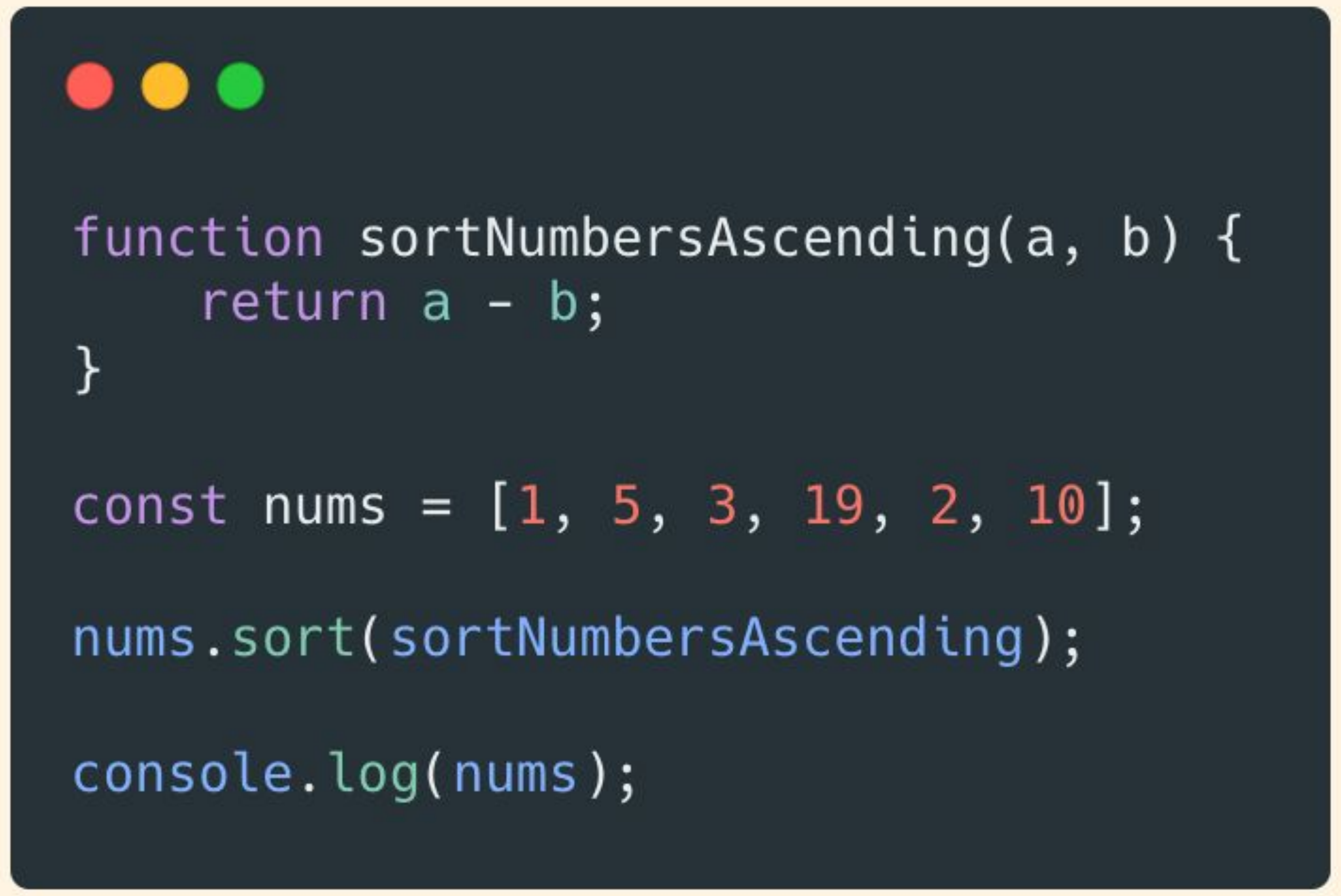
TASK

What do you think will happen if we add the `reverse()` after the `sort()`? I.e. `names.sort().reverse();`

Try it out

SORTING FUNCTION

`sort` takes the array and sorts the items in a kind of alphabetic order (but be careful of capital letters and special characters). `sort` can be (and usually is) customised to sort things in any way you like. It can take a function as an argument to tell it what to do.

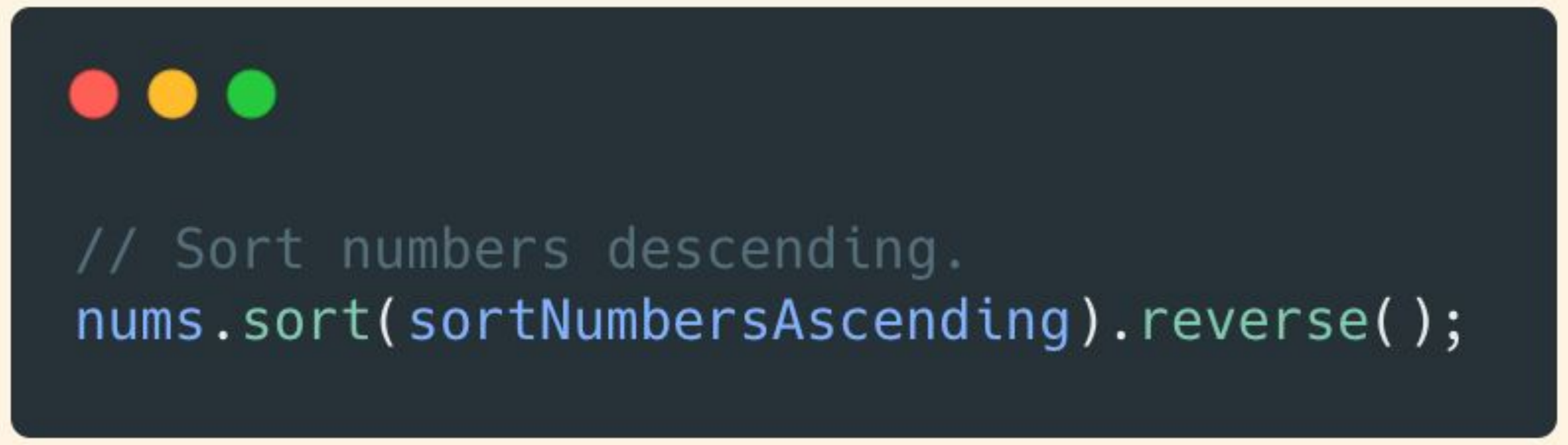


```
function sortNumbersAscending(a, b) {
    return a - b;
}

const nums = [1, 5, 3, 19, 2, 10];

nums.sort(sortNumbersAscending);

console.log(nums);
```

SORTING FUNCTION

BLACK CODHER

Sort passes pairs of entries from the array to `sortNumbersAscending`. If `sortNumbersAscending` returns a number less than zero, then sort knows that `a` should come before `b`. If the number is greater than zero, then `b` should come before `a`.



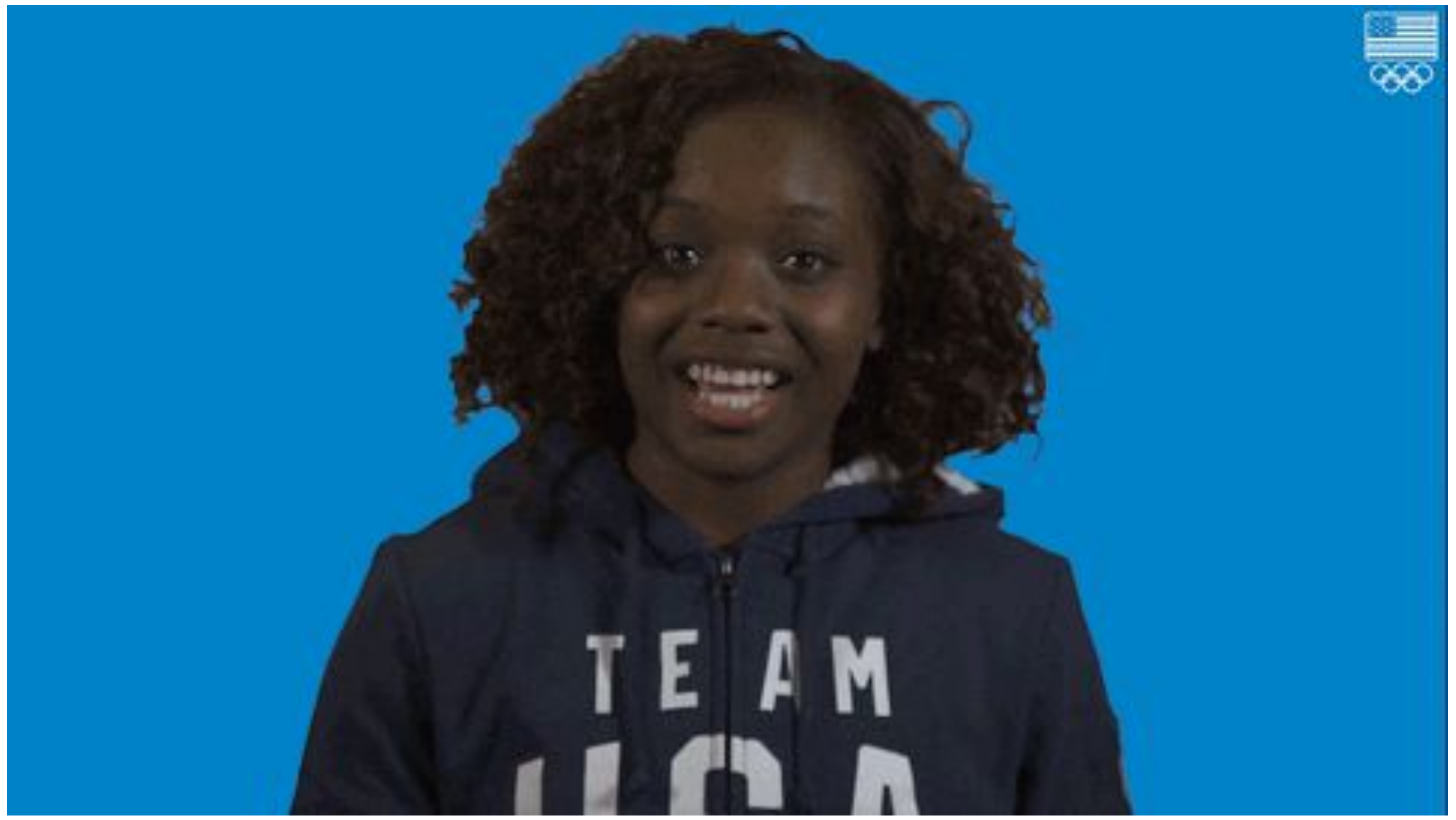
```
// Sort numbers descending.
nums.sort(sortNumbersAscending).reverse();
```

TASK

BLACK CODHER

Write a function called `sortNumbersDescending` that can be used in place of `.sort(sortNumbersAscending).reverse()`?

WHAT YOU'VE LEARNED SO FAR



- What an **array** is
- How we can use the `length` property in **for loops**
- How to sort elements in an **array**

Checkpoint!

How are you feeling?

RED - I have no idea what you're talking about

YELLOW - I have some questions but feel like I understand some things

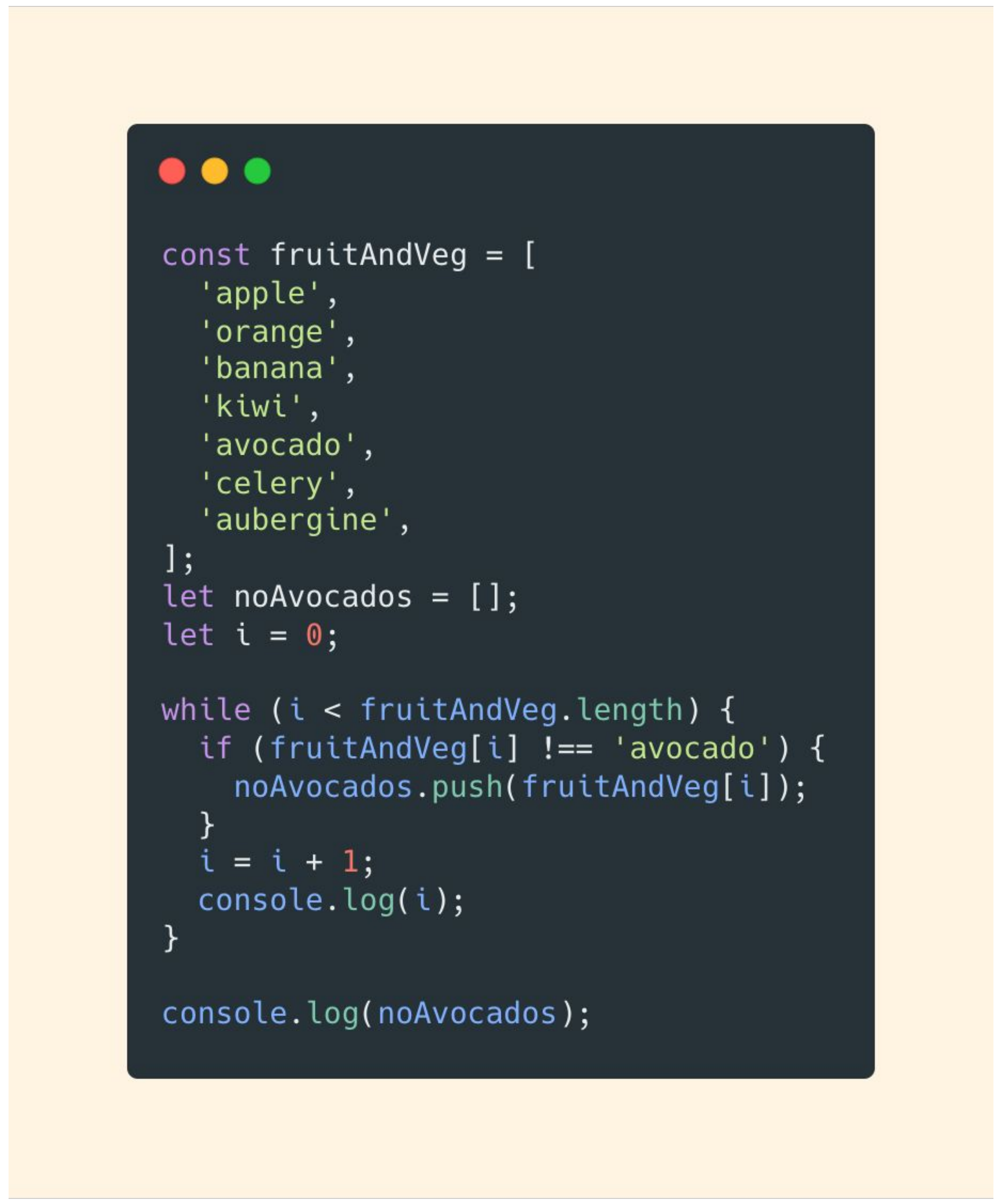
GREEN - I feel comfortable with everything you've said

GOING LOOPY!

LOOPS AGAIN

Now we know what arrays are, we can use them to further understand loops.

Example:



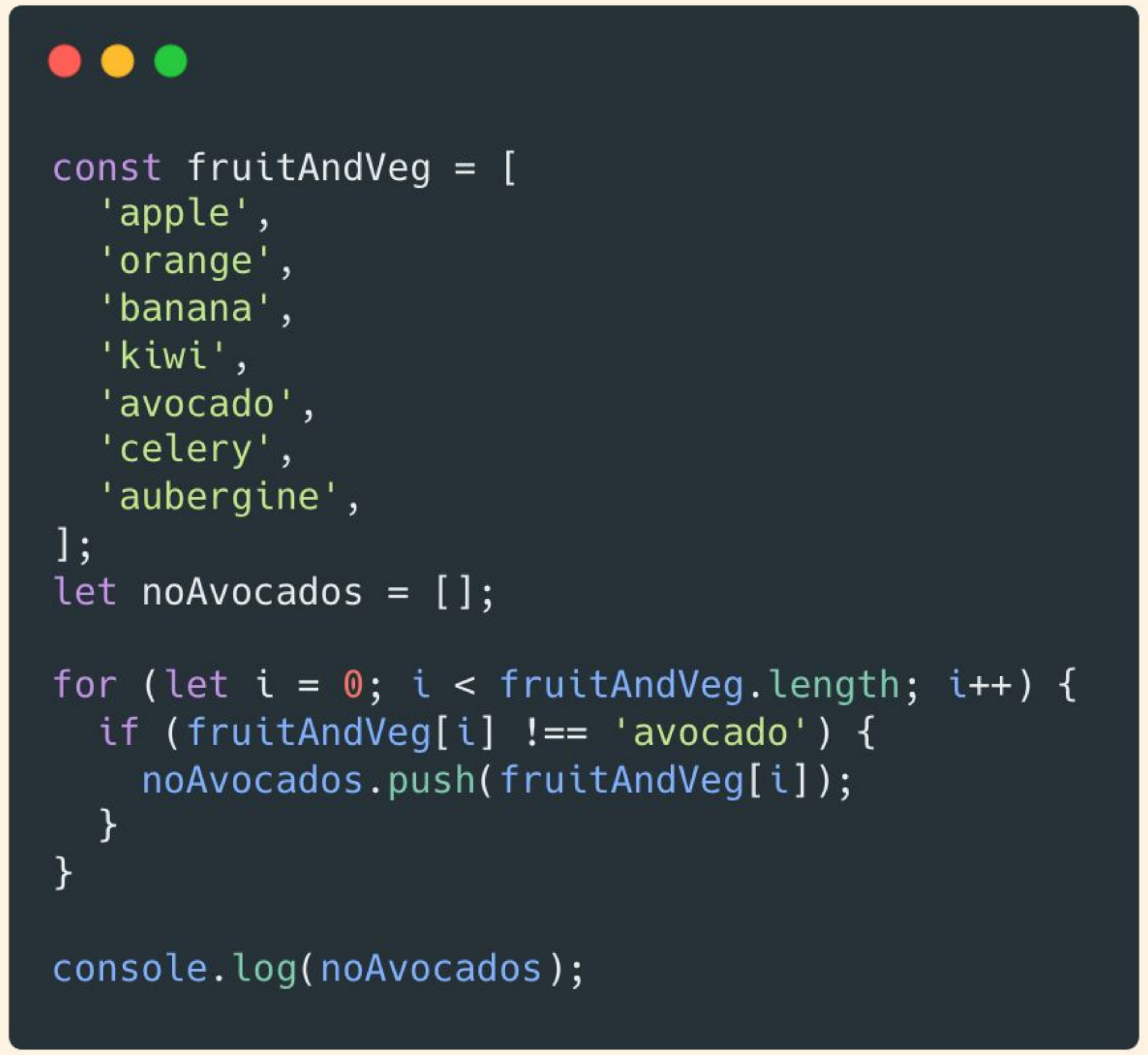
```
const fruitAndVeg = [
  'apple',
  'orange',
  'banana',
  'kiwi',
  'avocado',
  'celery',
  'aubergine',
];
let noAvocados = [];
let i = 0;

while (i < fruitAndVeg.length) {
  if (fruitAndVeg[i] !== 'avocado') {
    noAvocados.push(fruitAndVeg[i]);
  }
  i = i + 1;
  console.log(i);
}

console.log(noAvocados);
```

LET’S TALK SIS

Do you understand what this loop is doing? Try explaining it

LOOPS AGAIN

There is a better way to write this loop:



```
const fruitAndVeg = [
  'apple',
  'orange',
  'banana',
  'kiwi',
  'avocado',
  'celery',
  'aubergine',
];
let noAvocados = [];

for (let i = 0; i < fruitAndVeg.length; i++) {
  if (fruitAndVeg[i] !== 'avocado') {
    noAvocados.push(fruitAndVeg[i]);
  }
}

console.log(noAvocados);
```

LOOPS AND MORE ARRAY METHODS

In recent years new array methods have been introduced. These include:

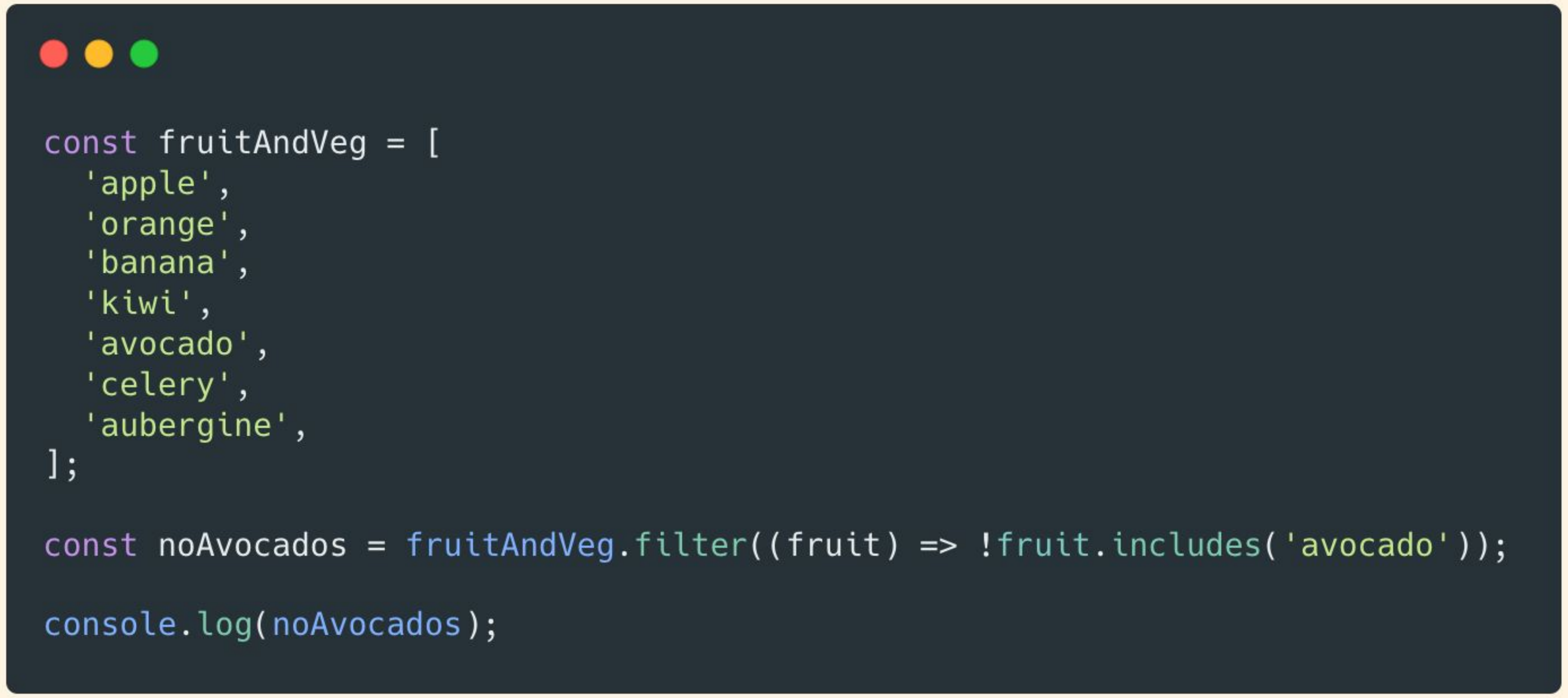
- `map`
- `filter`
- `find`

With these new methods we can **implicitly return** the results of our loops to a new variable (i.e. we create a new array of the results), have more **functional**-style code and use **arrow functions**.

Do you remember **arrow functions** from a previous session?

LOOPS AND MORE ARRAY METHODS

We can utilise the `filter` method and the string `includes` method to make the previous loop *even* better:



```
const fruitAndVeg = [
  'apple',
  'orange',
  'banana',
  'kiwi',
  'avocado',
  'celery',
  'aubergine',
];

const noAvocados = fruitAndVeg.filter((fruit) => !fruit.includes('avocado'));

console.log(noAvocados);
```

FILTER METHOD

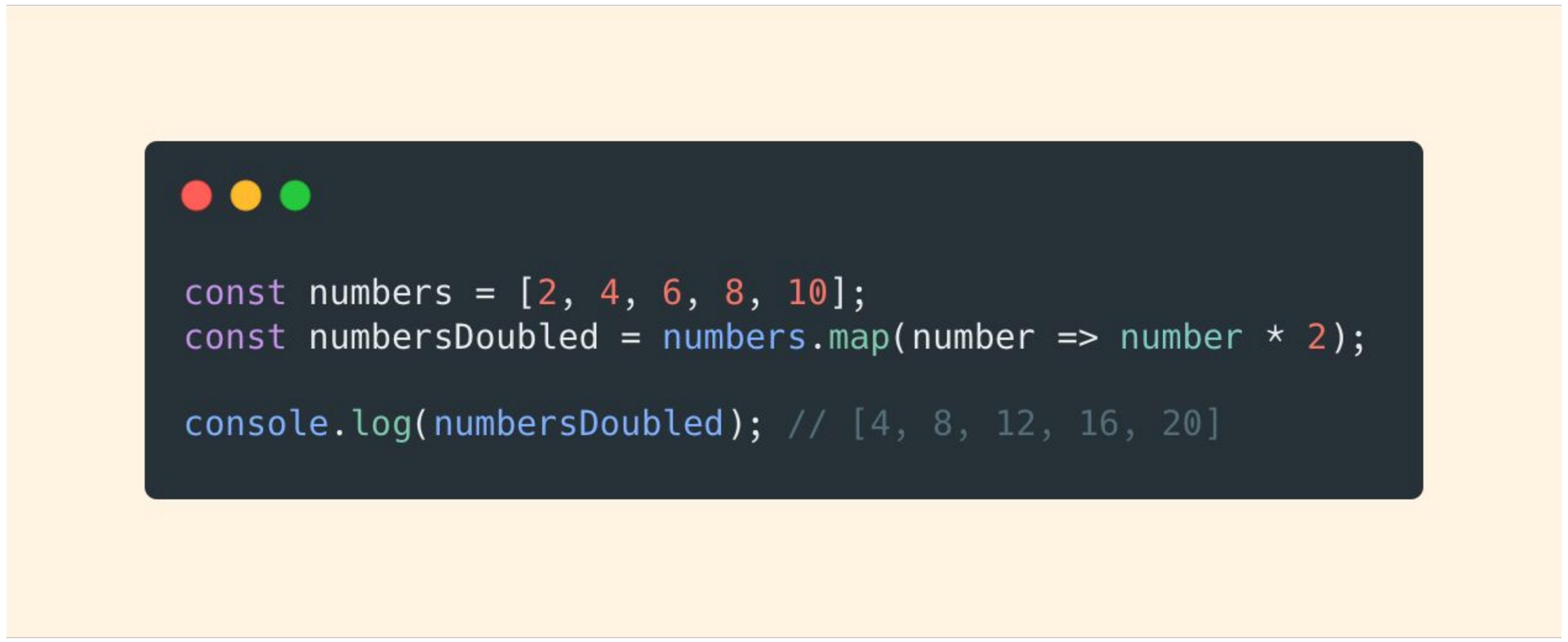
`filter` is a commonly used method that returns a new array and each element in the new array has passed our condition.

Using the previous example, in simpler terms, we are saying “*please give me an array of fruit and veg that doesn’t include avocados*”

MAP METHOD

`map` is another commonly used array method. It returns a new array containing the results of calling a function for every array element.

Let's return a new array that doubles the number in the original array.



```
const numbers = [2, 4, 6, 8, 10];
const numbersDoubled = numbers.map(number => number * 2);

console.log(numbersDoubled); // [4, 8, 12, 16, 20]
```

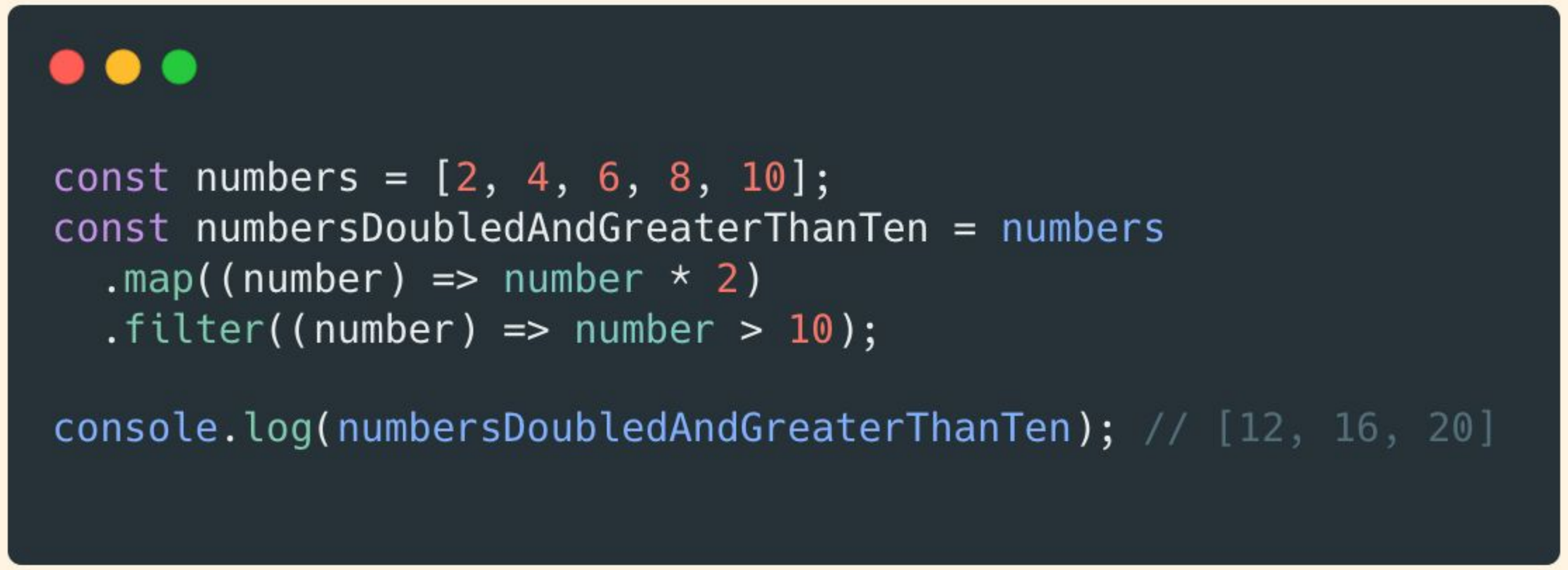
CHAINING METHODS

BLACK CODHER
CODING PROGRAMME

Because the newer array methods (e.g. `map` and `filter`) return brand new arrays you can call one array method directly after another.

This is commonly called **chaining** methods, because each call is like a link in a chain.

Let's add a `filter` to the previous example so our array only returns numbers greater than 10:



```javascript
const numbers = [2, 4, 6, 8, 10];
const numbersDoubledAndGreaterThanTen = numbers
  .map((number) => number * 2)
  .filter((number) => number > 10);

console.log(numbersDoubledAndGreaterThanTen); // [12, 16, 20]
```

TASK

BLACK CODHER

In the `index.js` file in the *session3* folder you will find an array of objects called `people`.

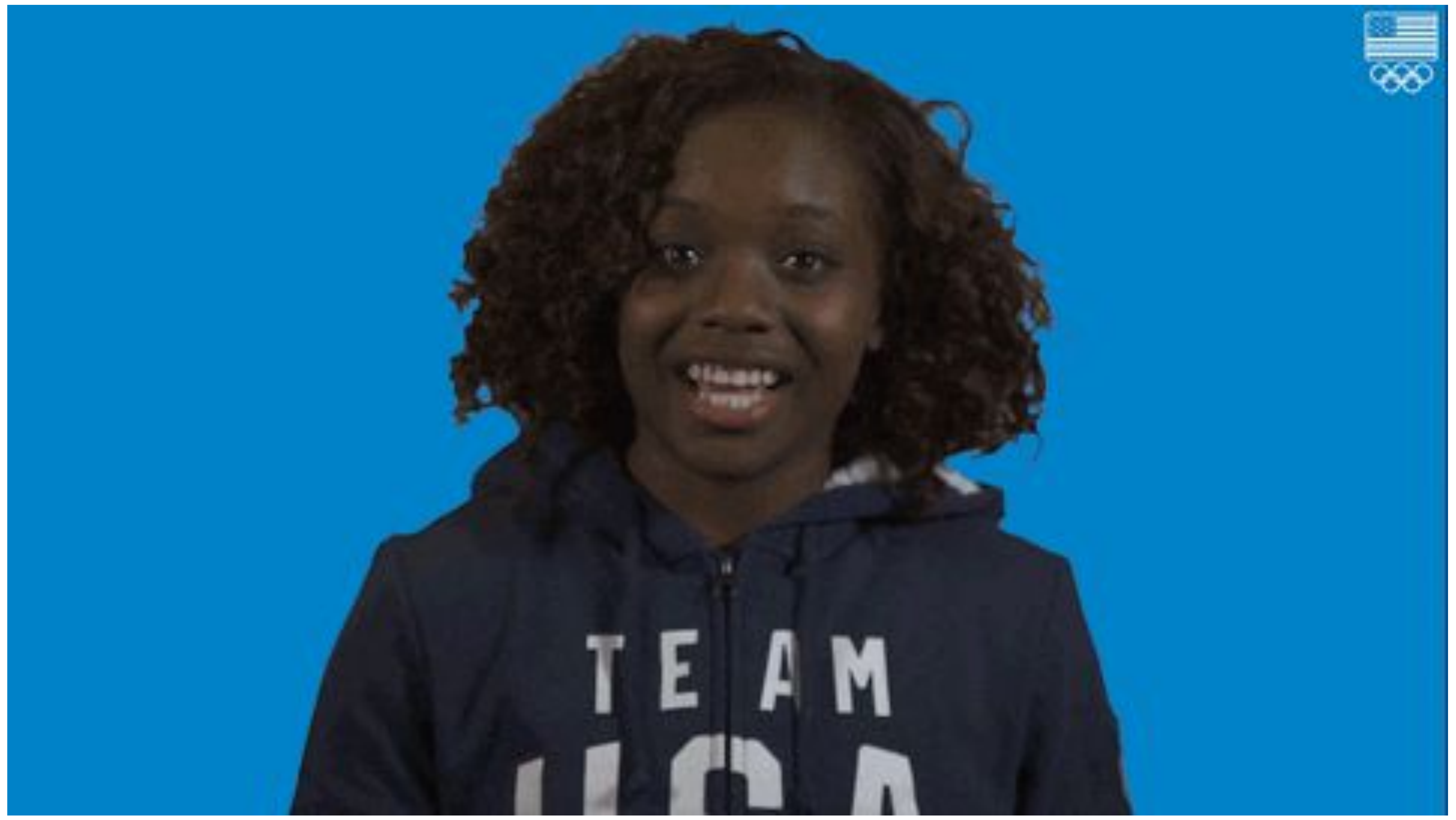
Your task is to return an array that only has the name of each person and this array should be sorted by their age (youngest to oldest).

Your final array should be:

`[“shuri”, “killmonger”, “t’challa”]`

There is already a `compare` function in the `index.js` ready for you to use.

WHAT YOU'VE LEARNED SO FAR



- What an **array** is
- How we can use the `length` property in **for loops**
- How to sort elements in an **array**

Checkpoint!

How are you feeling?

RED - I have no idea what you're talking about

YELLOW - I have some questions but feel like I understand some things

GREEN - I feel comfortable with everything you've said

SUMMARY

SUMMARY

- A `while` statement creates a loop that executes a block of code as long as the test condition evaluates to `true`.
- A `for` loop statement allows you to create a loop with three optional expressions.
- You should know when to use `for` loop instead of a `while` loop.
- An **array** is an ordered list of data.
- You should know how to use **arrays** and **loops** together
- There are various methods available to **arrays**.

WHAT'S NEXT?

Now we've learned the fundamentals of Javascript it's time to connect it to some HTML and finally begin manipulating the DOM.